TRAIT EXTRACTION FROM ARTICLE TEXT

By

Trace Braxling, B.S.

A Project Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

University of Alaska Fairbanks

May 2020

APPROVED:

Dr. Chris Hartman, Committee Chair
Dr. Jonathan Metzgar, Committee Member
Dr. Glenn G. Chappell, Committee Member
Dr. Chris Hartman, Department Chair
Department of Computer Science

Abstract

For a particular object, vast amounts of information exists within the Wikipedia article relating to that object. From this article, it is often desirable to produce simple, sentence length facts about this object.

The purpose of this project is to explore a number of methods for producing single sentences that provide informational facts (or traits) from a given text. These methods are then evaluated in comparison to each other, as well as a hand picked ground truth. By analyzing these results, it can be determined which aspects of the methods is worth further examination for the task of trait extraction.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank the faculty of the University of Alaska Fairbanks Computer Science Department. I learned a lot from their classes, and was helped countless times in any task I had. I would also like to thank the students of the same department for helping me think of new ways of solving work problems.

1 Introduction

Language understanding is an unsolved problem in computer science. One use of computers within language understanding is extracting information from text in various forms. This paper is concerned with finding *trait sentences*. A trait sentence is defined as a single sentence that that provides information about a particular thing, and make grammatical sense when alone. For example, the sentence

The cat (Felis catus) is a domestic species of small carnivorous mammal.

would be a valid trait sentence. In contrast, the sentence

Its night vision and sense of smell are well developed.

would not be a valid trait sentence, as it cannot be inferred what the sentence is referring to if it existed alone. The second sentence does not make sense without the context of the first sentence. This project concerns methods for creating trait sentences. The four methods are:

- Hand Construction
- Neural Network
- Regular Expression (Regex) Matching
- Part-Of-Speech (POS) tree

The hand construction method is a *manual* method used to obtain *trait patterns*. The other three methods are *automatic*, and use previously saved trait patterns to find trait sentences. Obtained trait patterns for a particular article can be used to find trait sentences for another article. Figure 1.1 shows the general form of trait sentence acquisition used by the automatic methods.

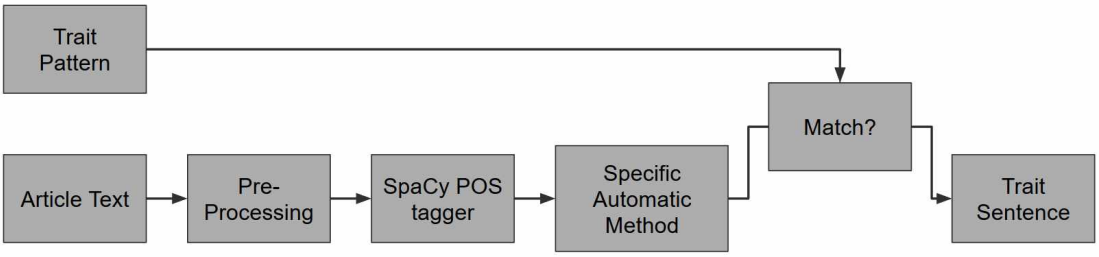


Figure 1.1: Trait Sentence Acquisition Process

2 Previous Work

Tanon et al. [12] states how "Collaborative knowledge bases are central for the data strategy of many projects and organizations". One use of these databases is providing resources for computers to generate sentences that state facts about a object. One such database is the Wikidata[14] database. With data in a machine-readable form, trait sentences can be produced. However, the amount of data available in Wikidata is much less then Wikipedia. For example, when comparing the Wikidata entry for *cat* to the same Wikipedia entry, it can be seen that Wikipedia has facts not present in the Wikidata entry. Wikipedia, however, cannot be easily converted to a machine-readable form to obtain trait sentences. Medelyan et al.[9] describe a method for obtaining facts using the structure and relations within Wikipedia itself. Using these concepts, various other tools, including the *Wikipedia Miner Toolkit* [10] have been developed, and allow developers to implement Wikipedia data mining techniques in their own applications. Recent developments projects include work done by Korn et al.[7]. In this, facts are generated using a table based approach where views are used to create fact sentences.

Our application does not rely on the inherent structure of Wikipedia. Due to this, it could be potentially modified for use on any plain text database.

3 Implementation

3.1 Code

The project is primarily implemented in Python 3.7.4. Operations include reading and writing to files, text manipulation, performing calculation for the automatic methods, etc. Javascript and HTML are used to create the front-end interface. This includes loading articles, choosing methods to run, hand constructing sentences, viewing results, etc.

3.2 SpaCy

SpaCy[4] is heavily used in this project. Anytime a sentence is converted is converted to its Part-Of-Speech token form, the SpaCy module is used. Spacy uses its entity recognition model to produce a prediction of the grammatical structure of text. This includes information such as POS tags and how words within a sentence are related to each other.

Table 3.1 shows the POS tags used with in the project, as defined within SpaCy specifications. Definitions are taken from the Universal Universal Dependencies (UD) standard[11].

POS	Meaning	Definition	Examples
ADJ	Adjective	Words that typically modify nouns and specify their properties or attributes.	*Beautiful*, *Brave*, *Small*

ADP	Adposition	Belongs to a closed set of items that occur before (preposition) or after (postposition) a complement composed of a noun phrase, noun, pronoun, or clause that functions as a noun phrase, and that form a single structure with the complement to express its grammatical and semantic relation to another unit within a clause.	*on*, *in*, *to*
ADV	Adverb	Words that typically modify verbs for such categories as time, place, direction or manner.	*gently*, *quite*, *then*
AUX	Auxiliary	A function word that accompanies the lexical verb of a verb phrase and expresses grammatical distinctions not carried by the lexical verb, such as person, number, tense, mood, aspect, voice or evidentiality.	*has*, *was*, *got*
CCONJ	Coordinating Conjunction	A conjunction placed between words, phrases, clauses, or sentences of equal rank.	*and*, *but*, *or*
DET	Determiner	Words that modify nouns or noun phrases and express the reference of the noun phrase in context.	*a*, *the*, *every*
INTJ	Interjection	A word that is used most often as an exclamation or part of an exclamation.	*ah!*, *yikes!*, *ouch*

NOUN	Noun	A part of speech typically denoting a person, place, thing, animal or idea.	*man*, *rabbit*, *piano*, *tree*
NUM	Numeral	A word, functioning most typically as a determiner, adjective or pronoun, that expresses a number and a relation to the number, such as quantity, sequence, frequency or fraction.	*3*, *five*, *6 million*
PART	Particle	Function words that must be associated with another word or phrase to impart meaning and that do not satisfy definitions of other universal parts of speech	*'s* (Possessive), *not* (Negation)
PRON	Pronoun	Words that substitute for nouns or noun phrases, whose meaning is recoverable from the linguistic or extralinguistic context.	*he*, *they*, *I*
PROPN	Proper Noun	A noun that is the name of a specific individual, place, or object.	*Bob*, *Mary*, *Anchorage*
PUNCT	Punctuation	Non-alphabetical characters and character groups used in many languages to delimit linguistic units in printed text.	*. , ( )*
SCONJ	Subordinating Conjunction	A conjunction that links constructions by making one of them a constituent of the other.	*if*, *while*
SYM	Symbol	A word-like entity that differs from ordinary words by form, function, or both.	*$*, *%*

VERB	Verb	A member of the syntactic class of words that typically signal events and actions, can constitute a minimal predicate in a clause, and govern the number and types of other constituents which may occur in the clause.	*jump*, *fly*, *swimming*
X	Other	Does not match any other tag	*aksbcsj*, *wwykq*
SPACE	Space	Literal Space	' '

Table 3.1: POS Tokens Used by SpaCy taken from the UD standard

The POS tagging featue of SpaCy is used extensively in the project, and is used in every method of trait sentence generation. For example, with SpaCy the sentence

The cat is a domestic species of small carnivorous mammal.

will produce the POS form of

DET NOUN AUX DET ADJ NOUN ADP ADJ ADJ NOUN

The SpaCy document class is used within the POS tree method to match sentences. This is due to the powerful sentence structure operations it provides.

3.3 Wikipedia-api

Wikipedia-api is used to pull Wikipedia articles. Given a article name, returns a document class. Features from this class that the project uses are returning text from individual or all sections, and finding the summary of an article.

3.4 Tensorflow

Tensorflow is used for its neural network computational features. It is used in the neural network method of the project.

3.5 NumPy

NumPy is used with Tensorflow to process large vector arrays and provide mathematical functions for these arrays.

3.6 jQuery

jQuery is used to return results to a front-end interface.

3.7 Source Code

All source code and data used for this project is available in a public version control repository [2].

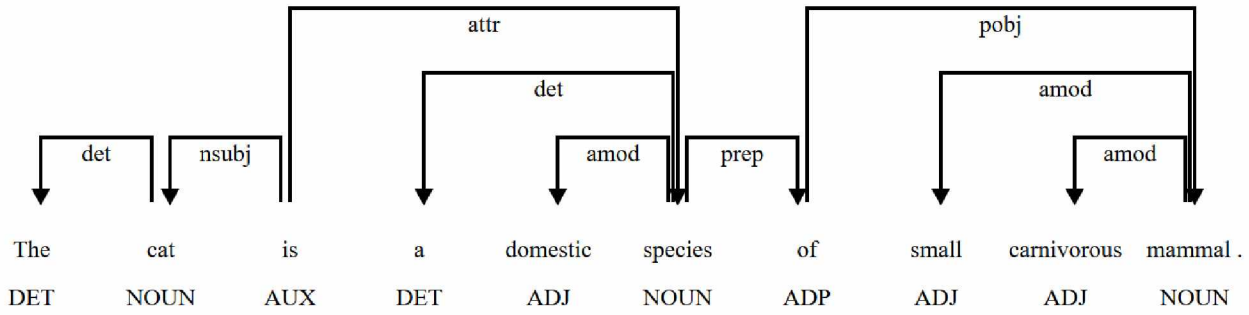


Figure 4.1: Dependency parse example for the sentence *The cat is a domestic species of small carnivorous mammal.*

4 Tool

The first method obtains trait sentences by providing a tool for users to construct them by hand. It is necessary to use the tool to create trait patterns, as the automatic methods will not produce trait sentences if there are no trait patterns. This means that this method must be used before the automatic methods. As for how it functions, the tool firstly allows the user to load an arbitrary Wikipedia document. This is accomplished through the use of the [8] Wikipediaapi module. After loading an article, each sentence is separated into a numbered list. The section heading for each article is also displayed.

When clicking on a sentence, it is displayed at the top of the page and separated into words. As well, the POS tag for each word is displayed. Sentences are POS tagged using [4] SpaCy's entity recognition model based on the work of Eli Klipperwasser and Yoav Goldberg[6], and the SyntaxNet[13] team from Google.

SpaCy also constructs a dependency parse[5] of a given sentence. This is a graphical analysis of a sentence that features directed, labeled arcs from heads to dependents. An example dependency parse is shown in Figure 4.1.

A text box is provided so a user can input any sentence to view its dependency parse if they wish. Often times, a sentence within an article can be reduced to a smaller sentence. If desired, this sentence can be manually created by clicking on one word at a time to create the user-constructed reduced sentence. For example, using this tool, the sentence:

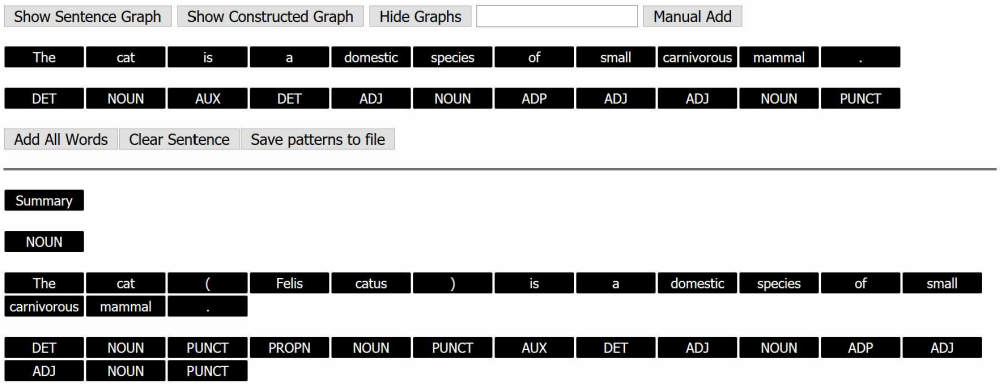


Figure 4.2: Using the tool to produce a user-constructed reduced sentence

The European rabbit, which has been introduced on every continent except Antarctica, is familiar throughout the world as a wild prey animal and as a domesticated form of livestock and pet.

Ccan be used to construct the sentence:

The European rabbit, is familiar throughout the world as a wild prey animal.

This reduced sentence can be used as a trait sentence, and a dependency parse of the user-constructed reduced sentence can also be displayed. An example of using the tool to perform this function is shown in Figure 4.2.

The tool can display results returned by any of the automatic methods. When an article is loaded, buttons to run each of the automatic methods appears. When the method has computed results, they will be returned to front end where they can be viewed. This is done via a jQuery call, so that the page does not need to be refreshed to return results.

5 Hand construction

After the user creates a trait sentence by hand, this sentence can be used to save a *Trait Pattern*. A single Trait Pattern includes:

- The original sentence
- The POS tags of the original sentence
- The user-constructed reduced sentence
- The POS tags of the user-constructed reduced sentence
- The indices of the user-constructed reduced sentence in relation to the original sentence
- A Regex pattern matching the POS tags of the original sentence
- A Regex pattern matching the POS tags of the user-constructed reduced sentence original sentence

For evaluation purposes, the correctness of the other methods is determined by comparison to the hand-picked values.

6 Text Pre-processing

Before the automatic methods can be used to obtain trait sentences, they go through some processing. All sentences within an article are added to a list. Wikipedia section headings are removed. Optionally, a list of words is provided, where sentences with these words will not be processes. By default, this list includes a number of *Negative Polarity Items*[1]. The reasoning for this is to prevent false positive results where possible. Words used to filter sentences by can be added or removed as needed. Afterwards, one of the three automatic methods can be used to obtain trait sentences.

7 Result Acquisition

To obtain data for analyzing, a number of arbitrary nouns were first chosen. These were used as the articles that data was obtained from. The hand construction method was used to construct slightly over 730 trait patterns. Afterwards, each automatic method was used on each article, with the trait patterns obtained from that particular article temporarily removed. These values are compared against the hand construction method. Results of the various methods are detailed in their associated section.

8 Neural Network

8.1 Method

The first method we used to obtain trait sentences is through the use of a neural network. The model we chose to use was Google's *Universal Sentence Encoder* model[3]. The input for this model is a sentence of arbitrary length, and the output is a vector of fixed size. This model was chosen because it is constructed in such a way that similar sentences will product similar outputs, and was designed for general use.

To find trait sentences, each sentence in a selected article is converted to its POS sentence, with each word replaced with a POS token. When executed, this method will input every POS sentence within the selected article into the model to output the vector. The output vector for each trait pattern is also determined. For each combination of article sentence and trait pattern, the inner product is calculated. The result will be a value from 0.0 to 1.0 that describes how similar two sentences are. The list of every combination and the resulting values between the two sentences is then saved and used for analyzing.

Trait sentence results from the neural network method are saved as a dictionary and returned to the front end. Listed attributes are shown in Table 8.1. An example obtained neural network result is shown in Figure 8.1.

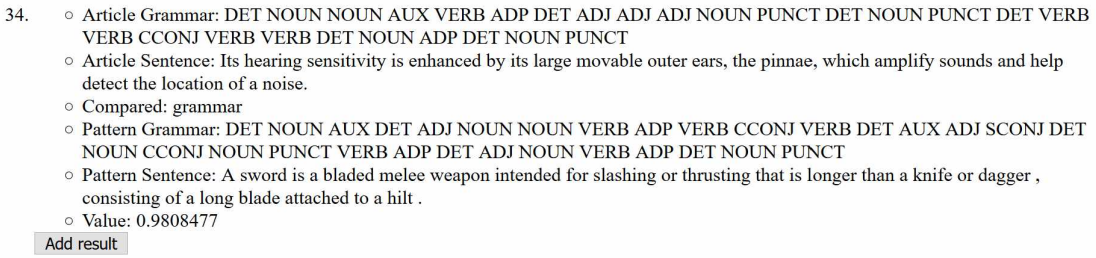


Figure 8.1: Neural Network Result Example

Attribute Name	Description
Article Sentence:	The sentence within the article that was matched to a trait pattern
Article Grammar:	The POS form of the article sentence
Pattern Sentence:	The trait pattern that matched to a sentence in the article
Pattern Grammar:	The POS form of the pattern sentence
Compared	What was compared within the neural network model • Grammar: POS form of sentences was input to model • Text: Actual text was input to model
Value:	The numeric value that was returned by the model

Table 8.1: Form of Neural Network Result

8.2 Results

To evaluate results for a particular article, the article is decoupled from trait patterns obtained from it. After the neural network method returned results, trait patterns were re-added. The list of trait patterns for that article were compared to the neural network results. At this point, it was noted whether or not results from the hand-picked list had results in the neural network list, and vice versa. Figure 8.2 shows the average neural network score of hand-picked values versus the average neural network score of values that were not hand-picked.

Figure 8.3 shows an example distribution of values for hand-picked and non hand-picked values in an arbitrary chosen article. Figure 8.4 shows this for another arbitrary chosen article. Note the small bump near the lower end of the graph. This was noted to be bibliography information.

Arbitrarily, sentence combinations with a value of 0.9 or greater were chosen as returned

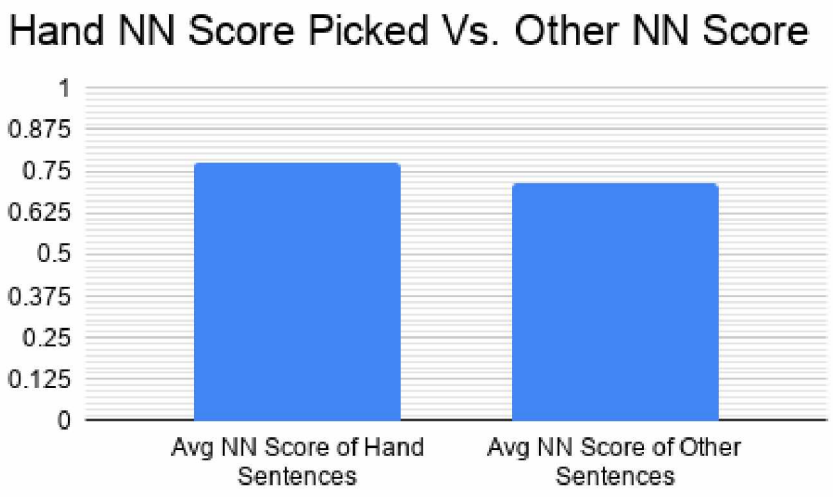
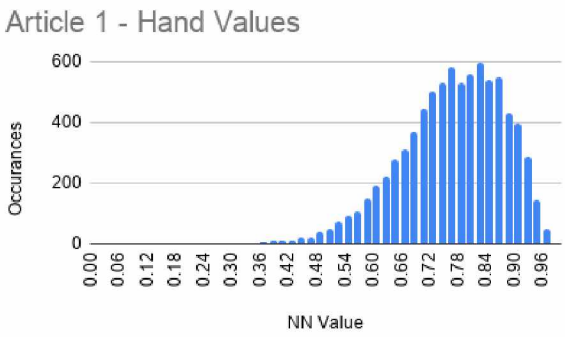


Figure 8.2: Dependency parse example for the sentence *Hand NN Score Picked Vs. Other NN Score*

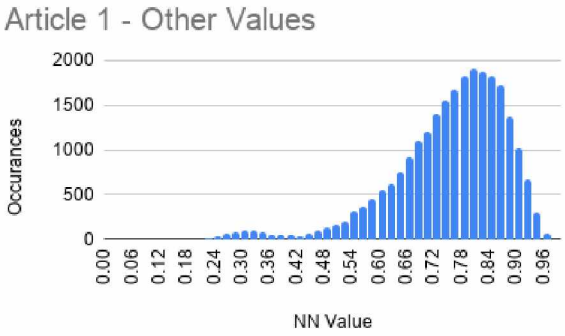
results for this method. These obtained trait sentences were compared against hand-picked trait sentences. The values recorded were then categorized based on the following criteria:

- Results in the hand-picked list but not in the neural network list was recorded as a *missed* trait sentence.
- Results that were present in both lists was recorded as a *found* trait sentence.
- Results in the neural network list but not in the hand-picked list was manually evaluated
  - If the result could be considered a trait sentence, it was recorded as an *extra* value missed when hand-picking trait sentences.
  - If the result was not a trait sentence, it was recorded as a *false positive*.

Figure 8.5 shows the average percentage of hand picked values found by the Neural Network tree method. Extra trait sentences not present in the original hand-picked list found by this method are noted. These sentences were then manually sorted to determine if they fit the criteria for being a trait sentence.



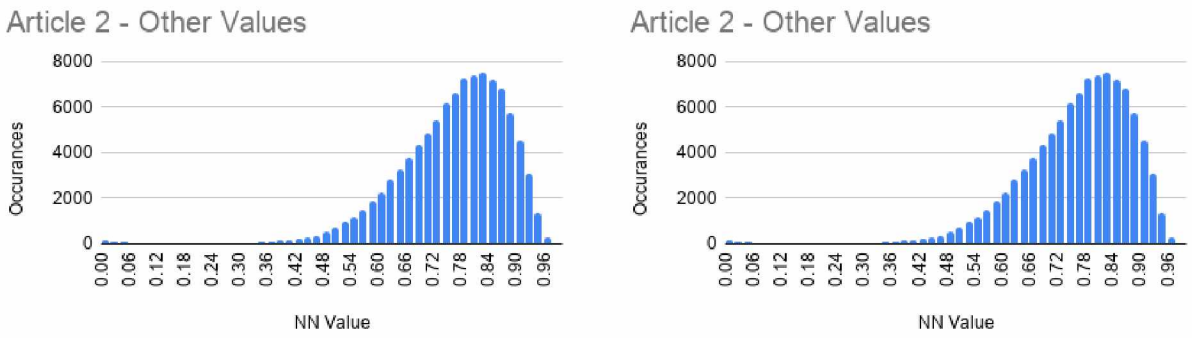
(a) Hand-picked Values



(b) Other Values

Figure 8.3: NN Distribution for Article 1

Figure 8.6 shows the average number of extra sentences found by this method, along with the manual designation of whether it does or does not fit the criteria for being a trait sentence. This same data is shown as a comparison to the number of hand-picked trait sentences in Figure 8.7.



(a) Hand-picked Values

(b) Other Values

Figure 8.4: NN Distribution for Article 2

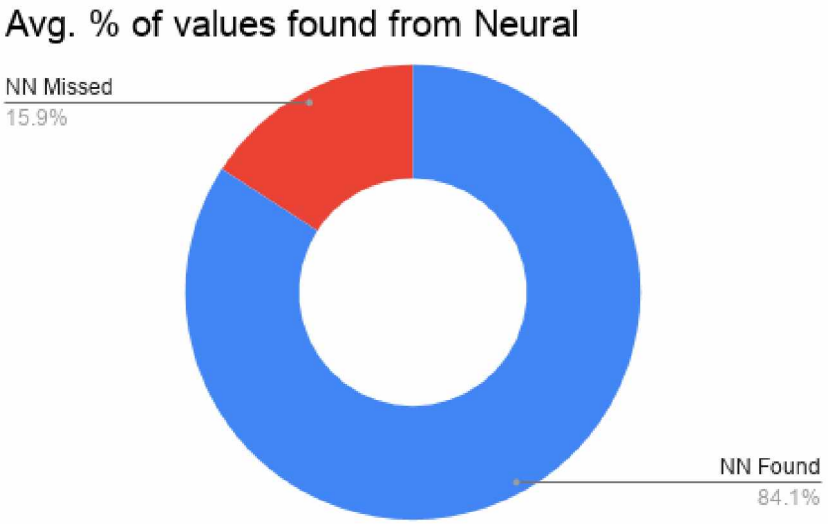


Figure 8.5: Average percent of values found from neural network method

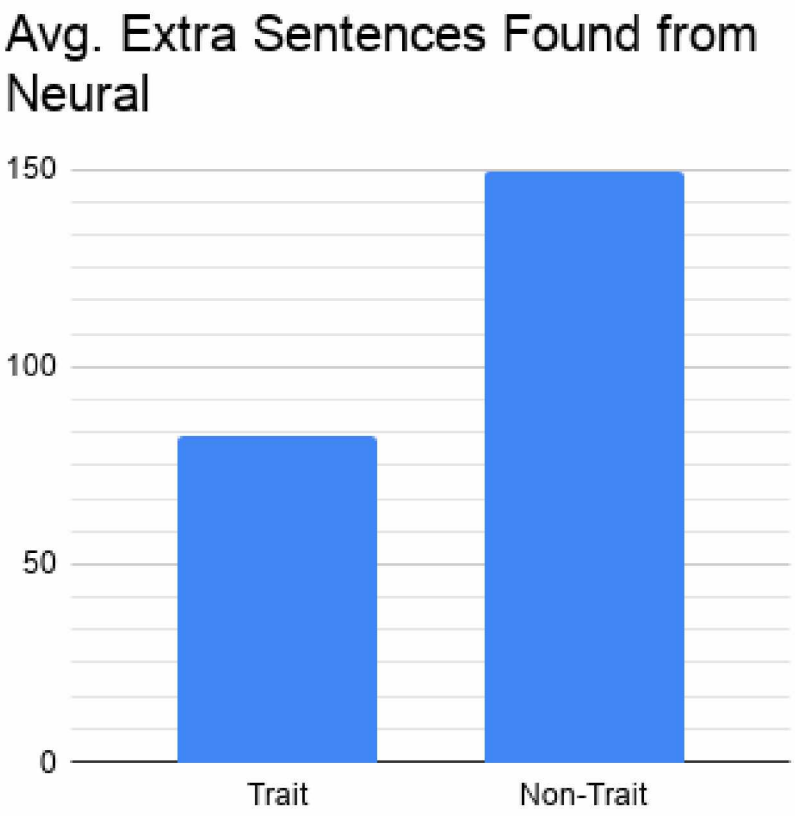


Figure 8.6: Extra sentences found by the neural network method

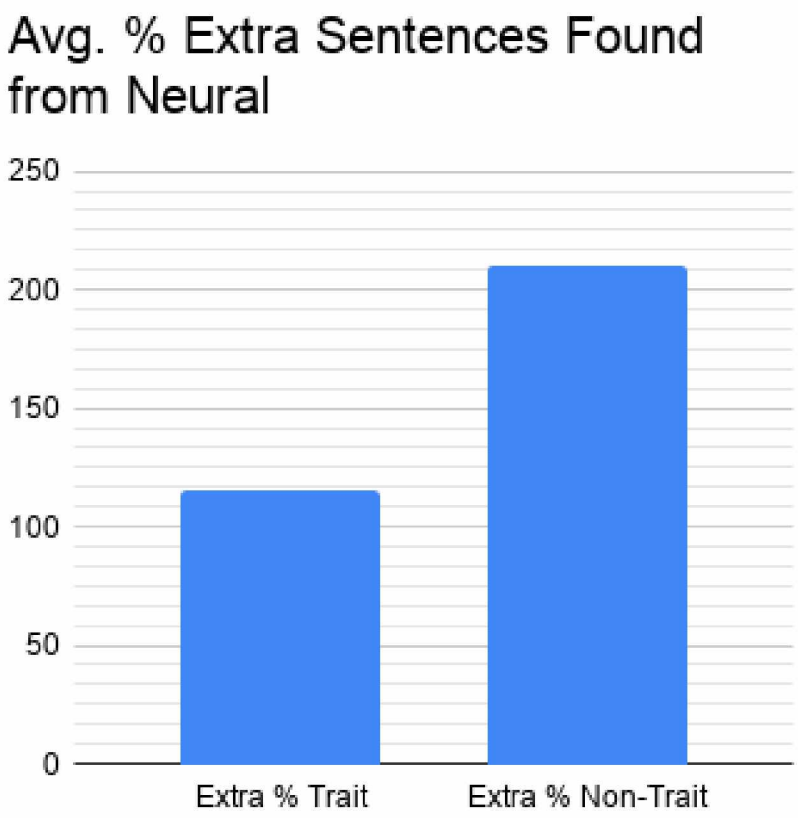


Figure 8.7: Percentage of extra sentences found by the neural network method

9 Regex Matching

9.1 Method

When a trait pattern is saved, a regex pattern is constructed. For this, a sentence is first converted into its POS form. Then, each POS token is saved as a capture group within the regex string. Between each token are wildcard characters with lazy evaluation. For example, the sentence

The cat is a domestic species of small carnivorous mammal.

will first be converted to its POS form

DET NOUN AUX DET ADJ NOUN ADP ADJ ADJ NOUN

Then, each token will be converted to a capture group, and lazy wildcard characters will be added between, resulting in the regex pattern

.?(DET).*?(NOUN).*?(AUX).*?(DET).*?-*
(ADJ).?(NOUN).*?(ADP).*?(ADJ).*?(ADJ).*?(NOUN)*

which will then be saved as part of the trait pattern.

To find trait sentences, each sentence within a given article is first converted to its POS form. Then, it is run against all saved regex trait patterns. Matches are saved and analyzed. Trait sentence results from the regex method are saved as a dictionary and returned to the front end. Listed attributes are shown in Table 9.1. An example obtained regex result is shown in Figure 9.1.

9.2 Reduced Sentence

One of the attributes returned is a reduced sentence that shows which words in the article sentence matched the regex capture groups.

For example, the regex pattern

Attribute Name	Description
Article Sentence:	The sentence within the article that was matched to a regex pattern
Article Tokens:	The article sentence converted to its POS form
Reduced Sentence:	The sentence constructed from the capture groups of the regex pattern
Regex Matched:	The regex pattern that matched to the article sentence

Table 9.1: Form of Regex Result

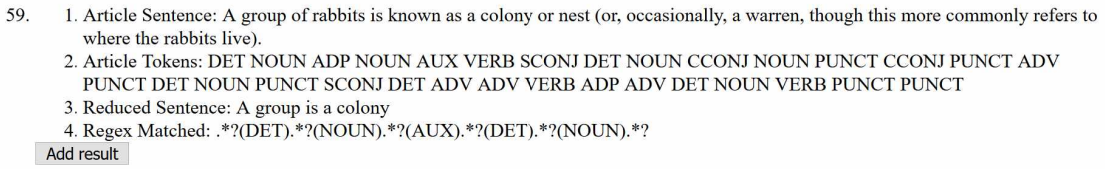


Figure 9.1: Regex Result Example

.?(DET).*?(NOUN).*?(AUX).*?(DET).*?(NOUN).*?*

would match the sentence

A goose (plural geese) is a bird of any of several waterfowl species in the family Anatidae.

This is because the POS form of this sentence is

DET NOUN PUNCT ADJ PROPN PUNCT AUX DET NOUN ADP DET ADP ADJ NOUN NOUN ADP DET NOUN PROPN PUNCT

Including only the matching terms would produce the reduced sentence

A goose is a bird

9.3 Results

To evaluate results for a particular article, the article is decoupled from trait patterns obtained from it. After the regex method returned results, trait patterns were re-added. The

list of trait patterns from that article were compared to the regex results. At this point, it was noted whether or not results from the hand-picked list had results in the regex list, and vice versa. The values recorded were then categorized based on the following criteria:

- Results in the hand-picked list but not in the regex list was recorded as a *missed* trait sentence.
- Results that were present in both lists was recorded as a *found* trait sentence.
- Results in the regex list but not in the hand-picked list was manually evaluated
  - If the result could be considered a trait sentence, it was recorded as an *extra* value missed when hand-picking trait sentences.
  - If the result was not a trait sentence, it was recorded as a *false positive*.

Figure 9.2 shows the average percentage of hand picked values found by the regex method.

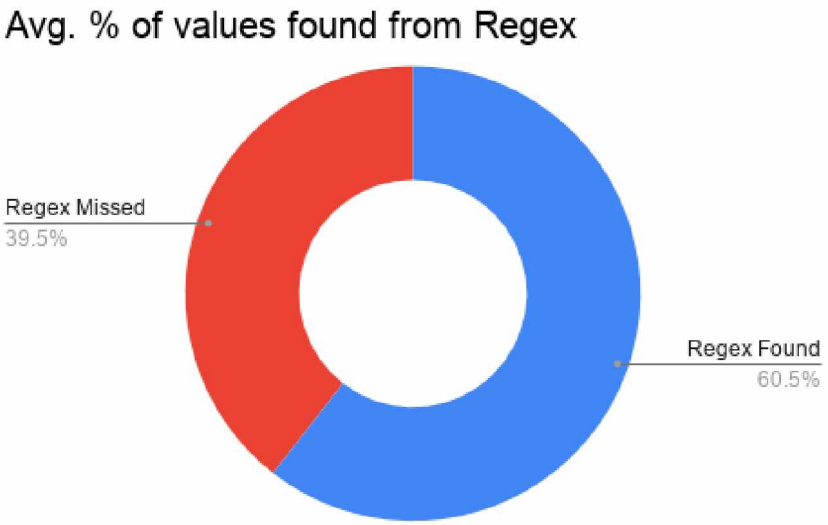


Figure 9.2: Average percent of values found from Regex method

Extra trait sentences not present in the original hand-picked list found by this method are noted. These sentences were then manually sorted to determine if they fit the criteria for being a trait sentence.

Figure 9.3 shows the average number of extra sentences found by this method, along with the manual designation of whether it does or does not fit the criteria for being a trait sentence. This same data is shown as a comparison to the number of hand-picked trait

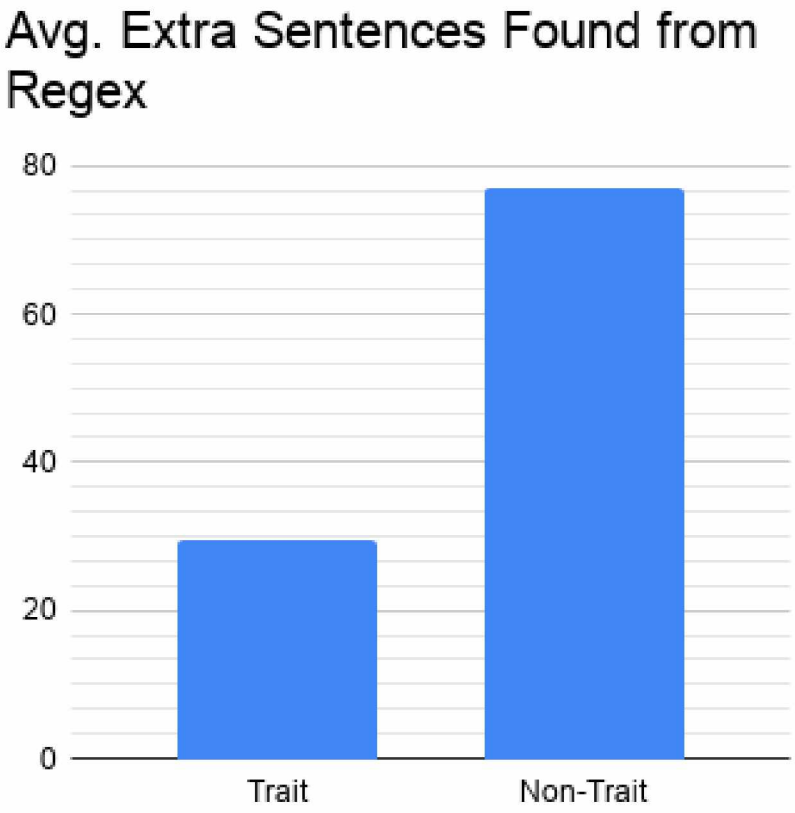


Figure 9.3: Extra sentences found by the regex method

sentences in Figure 9.4.

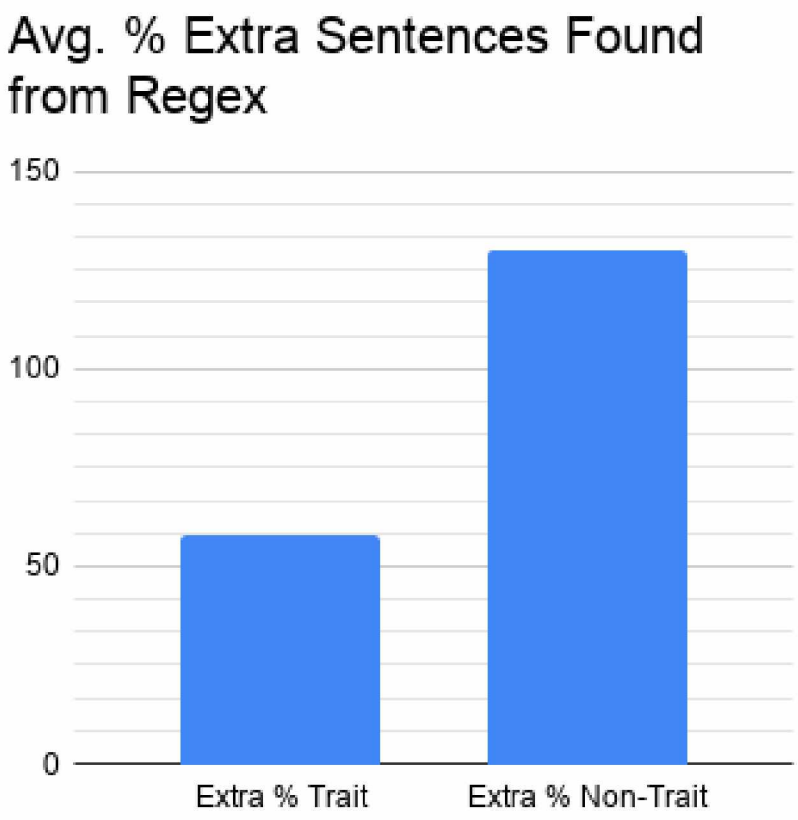


Figure 9.4: Percentage of extra sentences found by the regex method

10 POS Tree

10.1 Method

This method makes heavy use of Spacy's document class. To determine if an article sentence is a match within this method, a dependency parse of the sentence is construed. This was previously seen in Figure 4.1. A dependency parse of each trait pattern is then constructed.

To determine if two sentences match, the root token of each sentence is determined. From here, the algorithm than recursively checks the left and right dependency sub trees. For each step, if the POS tokens of the two sentences do not match, or there are missing or extra terms in the dependency parse, the match fails.

The exception is certain types of POS tokens, which there are allowed certain substitutions, additions, or deletions. For additions and deletions, this means that there can be certain POS tags present in one sentence, and missing in the other, and they will still match. For the project, the following POS tag addition and deletion rules were used:

- *Adjectives*
- *Coordinating conjunctions*
- *Punctuation*

Substitutions were implemented as lists of matching POS tags, where any tag in the same list could be substituted for each other. For example, the substitution list could be:

- (*POS TAG 0* , *POS TAG 1*)
- (*POS TAG 1*, *POS TAG 2* , *POS TAG 3*)

Meaning that *POS TAG 0* and *POS TAG 1* could be substituted for each other, or *POS TAG 1*, *POS TAG 2*, and *POS TAG 3* could be substituted for each other.

For the project, the following POS tag substitution rules were used:

- *Nouns* and *Proper nouns*

POS tokens can added to or removed to change the additions, deletions, or substitutions as needed.

Trait sentence results from the POS tree method are saved as a dictionary and returned to the front end. Listed attributes are shown in Table 10.1.

Attribute Name	Description
Article Sentence:	The sentence within the article that was matched to a trait pattern
Pattern Sentence:	The trait pattern that matched to a sentence in the article
Reduced Sentence:	The sentence constructed from removing certain POS tokens according to addition and deletion rules
Exact(Optional):	States if the POS tree is an exact match

Table 10.1: Form of POS tree Result

An example obtained POS tree result is shown in Figure 10.1.

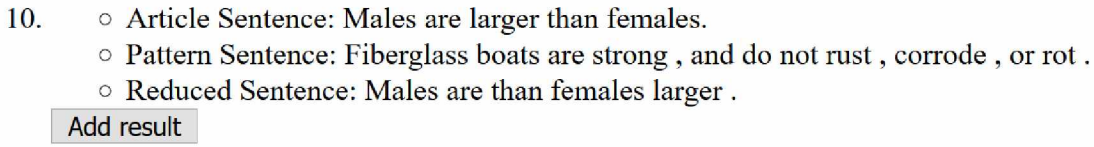


Figure 10.1: POS Tree Result Example

10.2 Reduced Sentence

One of the attributes returned is a reduced sentence that shows the article sentence with words removed according to the addition and deletion rules. All words that can be removed will be removed. For example, the sentence

Modern computers have billions or even trillions of bytes of memory.

With the form

ADJ NOUN AUX NOUN CCONJ ADV NOUN ADP NOUN ADP NOUN

Will be reduced to the sentence

Computers have billions or even trillions of bytes of memory.

10.3 Results

To evaluate results for a particular article, the article is decoupled from trait patterns obtained from it. After the POS tree method returned results, trait patterns were re-added. The list of trait patterns from that article were compared to the POS tree results. At this point, it was noted whether or not results from the hand-picked list had results in the POS tree list, and vice versa. The values recorded were then categorized based on the following criteria:

- Results in the hand-picked list but not in the POS tree list was recorded as a *missed* trait sentence.
- Results that were present in both lists was recorded as a *found* trait sentence.
- Results in POS tree list but not in the hand-picked list were manually evaluated
  - If the result could be considered a trait sentence, it was recorded as an *extra* value missed when hand-picking trait sentences.
  - If the result was not a trait sentence, it was recorded as a *false positive.*

Figure 10.2 shows the average percentage of hand picked values found by the POS tree method. Extra trait sentences not present in the original hand-picked list found by this method are noted. These sentences were then manually sorted to determine if they fit the criteria for being a trait sentence.

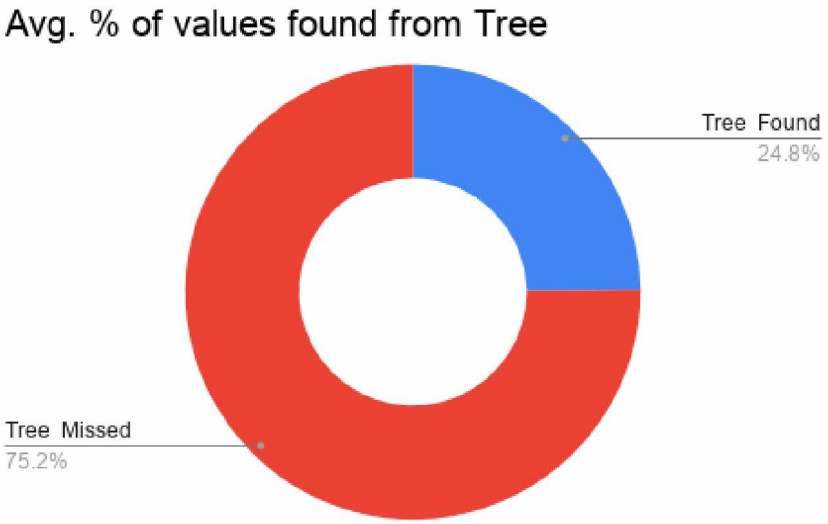


Figure 10.2: Average percent of values found from POS tree method

Figure 10.3 shows the average number of extra sentences found by this method, along with the manual designation of whether it does or does not fit the criteria for being a trait sentence. This same data is shown as a comparison to the number of hand-picked trait sentences in Figure 10.4.

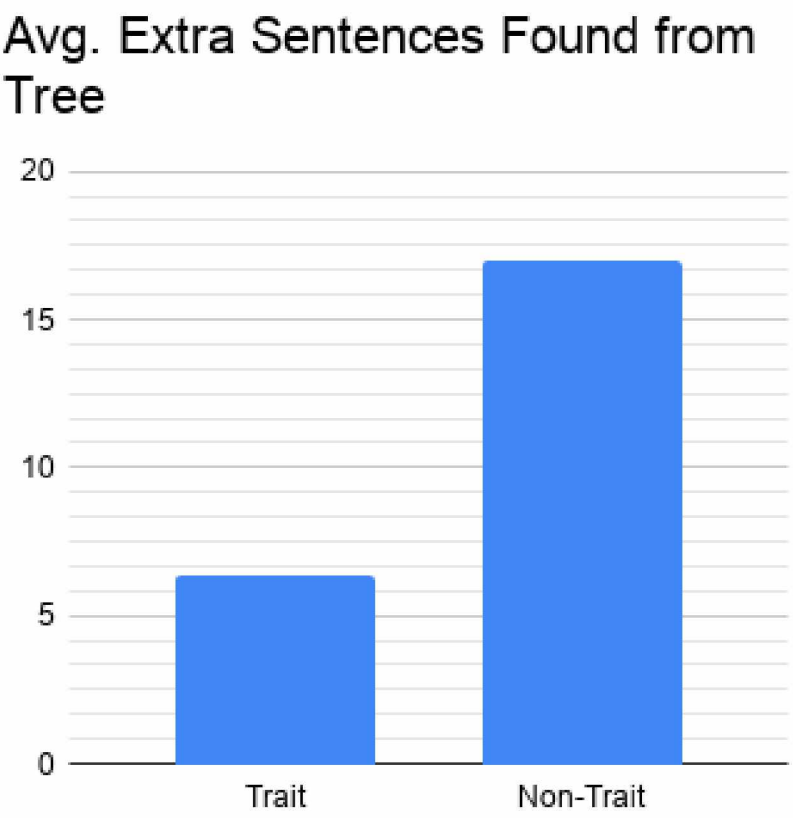


Figure 10.3: Extra sentences found by the tree method

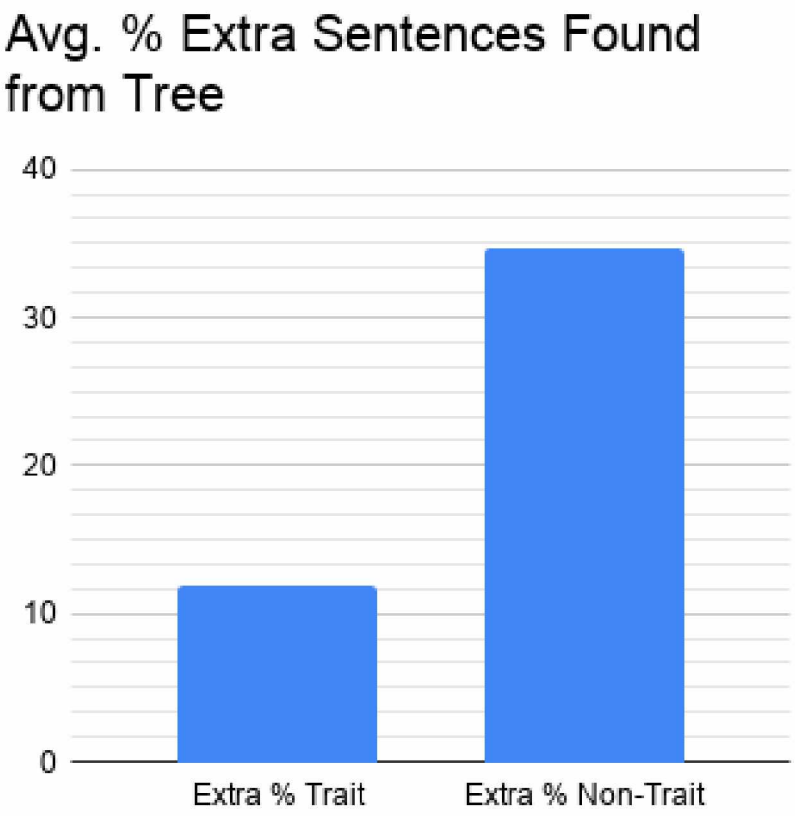


Figure 10.4: Percentage of extra sentences found by the tree method

11 Conclusion

Figure 11.1 compares the average percentage of hand picked values found by each method. Figure 11.2 compares the average number of extra sentences produced by each method.

In summarizing the results, it was determined that:

- The neural network method matched the largest number of trait sentences and had the largest number of false positives.
- The Regex method was between the other two automatic methods for trait sentences and false positives.
- The POS tree method matched the fewest trait sentences, but it also had the lowest number of false positives.

In conclusion, upon review of the three automatic trait sentence retrieval methods, we determined that the method with the most potential was the POS tree method. This was due to this method having the fewest false positive results. As well, this method has capability for further refinement as detailed in the next section.

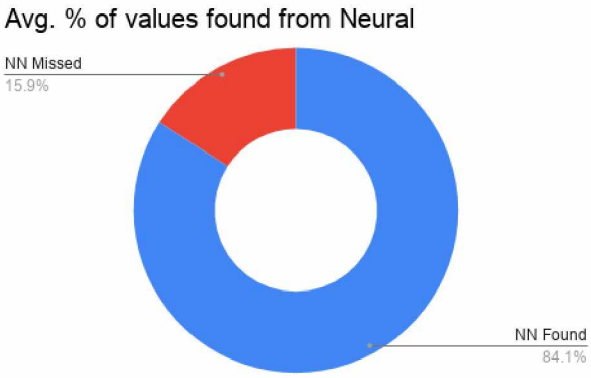
11.1 Future work

We determined that the POS tree method was the best automatic method for finding trait sentences. For future work, a more strict version of the POS tree method should be used. More elaborate rules can be added to determine if a POS token substitution, addition, or deletion should be allowed. As well, certain aspects of the other methods could supplement results provided by the POS tree method.

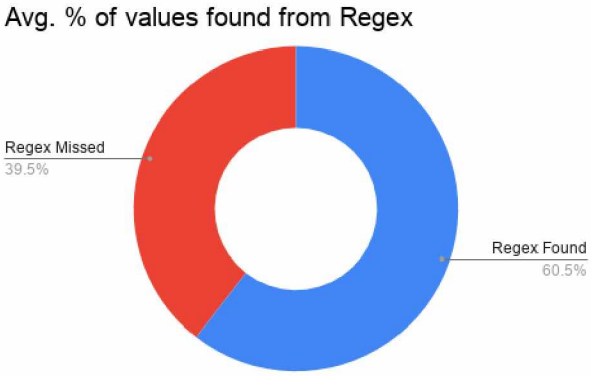
The regex method can be used to determine if a line of text is a proper sentence. This will reduce the occurrences of bibliography information and other malformed sentences being added to the list of sentences to operate on. More complex regex patterns could possibly be used to produce better results.

For the neural network method, very high scoring article sentences can be analyzed to determine if a sentence is close to matching a trait pattern. This sentence can then be inspected closer.

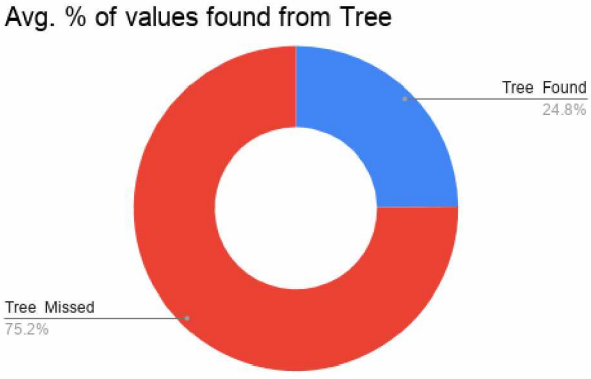
Finally, the hand construction method should be used to obtain more trait sentences to compare against, as more strict POS tree rules will require more trait patterns to match to article text.



(a) Neural

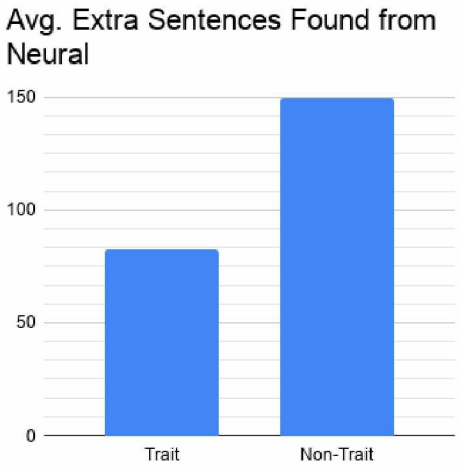


(b) Regex

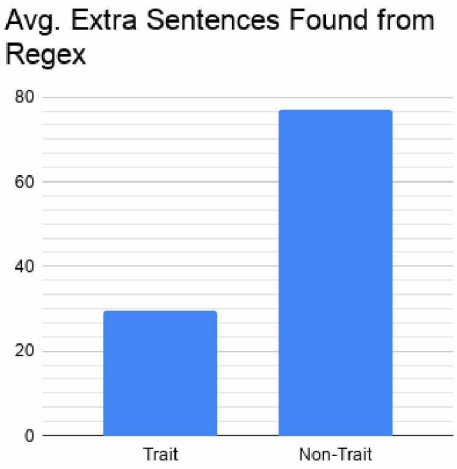


(c) POS Tree

Figure 11.1: Average percentage comparison



(a) Neural



(b) Regex

(c) POS Tree

Figure 11.2: Average extra sentence comparison

References

[1] Baker, C. L. (1970), Double negatives, *Linguistic inquiry*, *1*(2), 169–186.

[2] Braxling, T. (2020), Trait extraction from article text, `https://github.com/tbrax/informationPull`.

[3] Cer, D., Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. (2018), Universal sentence encoder, *arXiv preprint arXiv:1803.11175*.

[4] Honnibal, M., and I. Montani (2017), spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing, *To appear*, *7*(1).

[5] Keselj, V. (2009), Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, 2009, xxxi+ 988 pp; hardbound, isbn 978-0-13-187321-6.

[6] Kiperwasser, E., and Y. Goldberg (2016), Simple and accurate dependency parsing using bidirectional lstm feature representations, *Transactions of the Association for Computational Linguistics*, *4*, 313–327.

[7] Korn, F., X. Wang, Y. Wu, and C. Yu (2019), Automatically generating interesting facts from wikipedia tables, in *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, p. 349–361, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3299869.3314043.

[8] Majlis, M. (2017), Wikipedia-api.

[9] Medelyan, O., D. Milne, C. Legg, and I. H. Witten (2009), Mining meaning from wikipedia, *International Journal of Human-Computer Studies*, *67*(9), 716 – 754, doi: https://doi.org/10.1016/j.ijhcs.2009.05.004.

[10] Milne, D., and I. H. Witten (2013), An open-source toolkit for mining wikipedia, *Artificial Intelligence*, *194*, 222 – 239, doi:https://doi.org/10.1016/j.artint.2012.06.007, artificial Intelligence, Wikipedia and Semi-Structured Resources.

[11] Nivre, J., M.-C. De Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, et al. (2016), Universal dependencies v1: A multilingual treebank collection, in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pp. 1659–1666.

[12] Pellissier Tanon, T., D. Vrandečiundefined, S. Schaffert, T. Steiner, and L. Pintscher (2016), From freebase to wikidata: The great migration, in *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, p. 1419–1428, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, doi:10.1145/2872427.2874809.

[13] Petrov, S. (2016), Announcing syntaxnet: The world's most accurate parser goes open source, *Google Research Blog*, *12*.

[14] Vrandečić, D., and M. Krötzsch (2014), Wikidata: a free collaborative knowledgebase, *Communications of the ACM*, *57*(10), 78–85.